Summary (Unmarked are exam questions)

Operating systems (Technische Universiteit Eindhoven)

Introductions to Operating systems

Motivations existence of operating systems:

- Abstraction → provide useful generic concepts to handle complexity
- Virtualization(/Transparency/Deal with diversity)→ abstract model for wide range of systems
- Resource management → resource sharing, optimizing the performance
- Shared functionality →provide functionality for most common programs
- Concurrency → The machine must be shared
- Portability → support source code portability

Ways (mechanisms) that gives control to the operating system kernel:

- Trap
- Interrupt
- Exception

What are the steps in processing a system call?

- 1-3: pushing parameters
- 4: call library function
- 5: put code for read in reg.
- 6: trap: switch mode and call particular handler
- 7: handler calls read function handler
- 8: handler performs read actions (a.o. store data at address)
- 9-11: control back to caller

What is system availability?

- the probability that a system is functioning (according to specifications)
- or: the degree to which a system or component is operational and accessible when required for use (IEEE)

Large collection of services needed by virtually all programs:

- User interface (UI)
  - e.g. Command-Line (CLI), Graphical User Interface (GUI), Batch
- Program execution
  - load a program into MM and run
  - end execution, either normally or abnormally(indicating error)
- I/O operations
- File-system management
- Communication between processes
  - e.g. via shared memory or through message passing (packets)
- Error detection
  - Debugging facilities
- Resource sharing & allocation
  - CPU cycles, main memory, file storage, I/O devices (using timers)
- Accounting
  - Who's using which resources for how long?
- Protection
  - concurrent processes should not interfere with each other (memory)
- Security
  - of the system against running processes and outsiders

- Policy: Defines what you want a system to do
- Mechanism: defines how to do it
- Transparency: Hide details with respect to a given issue

Types of system calls:
- Process management
  - create, destroy, communication, synchronization, ...
- File management
  - open, close, read, write, ...
- Memory management
  - allocation, free, virtual memory, ...
- Device management
  - access control, open, attach, send/receive, ...
- Communications
  - setup communications, exchange messages, ...
- Miscellaneous
  - timers, inspect system resources, ...

Processes, threads and scheduling

What is scheduling, and what is a schedule?
- Scheduling is to allocate resources to tasks (processes or threads), which includes decision-making policies to determine the order in which active processes should compete for the use of resources and the actual binding of a selected task to a resource.
- Schedule is a function that maps a time and a resource to a task.

Under what circumstances is busy waiting acceptable? And where is it applied?
- When the waiting time is guaranteed to be short. Typical in synchronization between processors in critical sections.
- When it is the only thing that may happen on that processor. Typical in special purpose hardware for devices.

In the lectures, we discussed three fundamental aspects in which various CPU scheduling frameworks may differ from each other. List those aspects and explain each one briefly.
- When to schedule: When to activate the scheduler and go into decision mode?
- Who to schedule: What is the priority function for currently active processes? The highest priority process is scheduled next.
- What arbitration rule: Who to schedule in case of equal priority?

What is a task attribute, and what is a scheduling metric? Give an example of both.
- Attribute is a property a task has. It can be static or dynamic. Examples: arrival time, computation time, required resource, deadline, etc.
- A scheduling metric is an observed property. It is the result of applying scheduling policies on a task. Examples: response time, turnaround time, average overshoot, etc.

Some questions with fork() that ask you how many processes there are in the end. Example:

- Below is the code for a program named Agent_Smith.c. Including the initial parent process, how many Agent_Smith processes are created? Assume there are no errors. Show your work.

```
#include <stdio.h>
#include <unistd.h>
...
int main()
{
    pid_t smith;
    smith = fork( );

    if (smith == 0) {
        fork( ); fork( ); fork(); /* BEWARE */
    }
    else if (smith > 0) {
        fork( ); fork( );   /* BEWARE */
    }

    /* Consume resources of another process of the program Neo.c */
    /* This does NOT create a new process.                       */
    Consume( );   Consume( );

    return 0;
}
```

Answer: 12

What is 'priority inversion'? Give an example.

- The situation in scheduling that a high priority process is blocked by a process of middle priority, through blocking on a resource shared between this high and a (third) low priority process. During this blocking, arbitrary processes of middle priority can overtake. An example is found on the slides.

Mention the two basic concepts for inter-process communication and briefly describe them.

- Shared memory: memory space reserved by the kernel, addressable by both processes.
- Message passing: communication channels between processes (ex. pipes, sockets).

What is the difference between a mode switch and a context switch?

- mode switch: kernel ←→ user mode. Takes very little time.
- context switch: happens for process switch - stores the state of a process while restoring the state of another process (costs time).

A process includes:

- program code (text)
- stack
- heap
- data
- program counter

instructions executed one by one

Threads:

- User or Kernel threads are mapped to each other by the following mapping models:
  - Many-to-one
    - No need for kernel involvement
    - Multiple threads cannot run concurrently on different processors
    - A blocking system call for one user thread blocks all user threads of the process
  - One-to-one
    - Low performance in case of many user threads
  - Many-to-many
    - Allows the OS to create a limited number of kernel threads such that the number of user treads is still bigger than the amount of kernel threads
    - Concurrency level is limited by the number of kernel threads
  - Two-level model
    - Similar to many to many except that it allows a user thread to be bound to a kernel thread
- These mappings sometimes give mapping issues, solved by:
  - LWP: a user level representation of a kernel thread (like a virtual processor)
  - User threads are scheduled onto an available LWP
    - Blocking calls will switch LWP to a new user thread
  - Level of concurrency is the number of kernel threads
- Concurrency on memory address space of the process
- State per thread
- Generally little overhead for switching, switching has two possibilities:
  - Switching as a kernel activity (kernel maintains execution of state of thread)
  - Switching handled by a user level thread package (library) (and maintains state)
- Thread pools (a number of threads waiting for work)
  - Advantage: usually slightly faster to service a request with an existing thread than creating a new thread
  - Allows the number of threads in the applications to be bound by the size of the pool

A process is identified by an ID and a pointer to a PCB in kernel space, a PCB contains:

- Process state (see next slide)
- Process number
- Program counter (address of next instruction)
- CPU registers (needed for context switching)
- Memory management info (values of base, limit registers)
- I/O status information (list of open files, I/O devices)
- Scheduling information (priority, scheduling parameters)
- Accounting information (e.g. CPU time used)
- and more…

Process state:

- new: process is being created
- ready: process is waiting to be assigned to a CPU (before scheduler dispatch)
- running: instructions are being executed
- waiting: process is waiting for some event to occur
- terminated: process has finished execution

Scheduler can be invoked (decision mode), e.g. when a process:

1. switches from running to waiting state (e.g. wait for child to terminate)
2. switches from running to ready state (e.g. through an interrupt)
3. switches from waiting to ready (e.g. i/o completion)
4. terminates

If scheduling takes place ONLY under 1 & 4 (active process voluntarily yields)

➔ non-preemptive decision mode otherwise

➔ preemptive decision mode

Schedule algorithms:

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Priority Scheduling
- Round Robin scheduling (every time quantum of length q -> reschedule)
- Multilevel queue scheduling
- Rate monotonic scheduling (RM) (Real time: a process with period T is activated every T time units, and its computation must complete within a relative deadline of D time units.)
    - Decision mode: Preemptive
    - Priority function: Shorter T → Higher priority
    - Arbitration rule: chronical or random ordering
- Earliest Deadline first (EDF) (Highest priority is assigned to the process with the smallest remaining time until its deadline)
    - Decision mode: Preemptive
    - Priority function: least time remaining until deadline
    - Arbitration rule: chronical or random ordering

Concurrency

(Atomicity and interference, Action synchronization, Condition synchronization, Deadlocks)

What is interference of concurrent programs?

- The truth of an assertion that on local reasoning would be true is falsified.

What is meant by interference among concurrent threads?

- Assumptions or knowledge that one thread has about the state are disturbed by actions of another thread

Give the steps that make up a conditional critical region.

1. lock
2. while not condition do wait od
3. critical section
4. possible signals
5. unlock

A spinlock is a common approach to implementing mutual exclusion. What are its advantages and disadvantages, and where is it used?

- Advantage: work with multiprocessors.
- Disadvantage: busy waiting, cost time, doesn't work for a single processor.

Give two motivations for making a program multi-threaded.

- follow solution structure: natural concurrency
- take advantage of underlying platform concurrency
- hide latency

Consider the following statements in a language like C or Pascal:

```
x := y;
x := 5+z;
```

Here, x and y are shared variables and z is a local variable (accessible by just one thread). May these statements be regarded as atomic? Motivate your answer in one line.

- first assignment contains two references to shared variables: not atomic
- second assignment contains just one reference to shared variable: atomic

Mention the four correctness concerns in concurrent programs or systems.

- functional correctness
  - satisfy the given specification (e.g., mutual exclusion). An assertion that is needed for correctness must not be disturbed (by another process).
- minimal waiting
  - waiting only when correctness is in danger.
- absence of deadlock
  - don't manoeuvre (part of) the system into a state such that progress is no longer possible.
- fairness
  - (weak) eventually, each contender should be admitted to proceed.
  - (strong) we can put a bound on the waiting time of a contender.
  - absence of fairness: leads to starvation of processes.
- Absence of livelock
  - ensure convergence towards a decision in a synchronization protocol

What are the two principles of condition synchronization?

- At places where the truth of a condition is required: check and block
- At places where a condition may have become true: signal waiters

What is meant with the term 'race condition':

- The situation that correctness of a concurrent program depends on specific interleavings being chosen. For example, in a program that assumes that the state is not changed between a test and an action that depends on the result of this test.

What is the difference between the monitor signaling disciplines 'signal-and-wait' and 'signal-and-continue'?

- signal-and-wait: signaled process gets immediate access; signaler waits
- signal-and-continue: signaled process queues again for critical section access; signaler continues

When can we call a set of tasks D a deadlocked set?

- D is said to be deadlocked when D contains at least one not terminated task;
- all tasks in D are blocked
- for each non-terminated task t in D, any task that might unblock it is also in D.

Give three approaches to deal with deadlock.

- avoidance
- prevention
    - dynamic checks (expensive)
- detection
    - possible roll-back

Give 3 rules of thumb to avoid deadlock in action synchronization.

- Let critical sections terminate
- avoid greediness
- avoid cycles in semaphore calls.

In Posix, the value of a semaphore can be inspected. Is this useful in a program? If yes, explain with an example.

- Yes, You can use it to avoid blocking of a process, for example by inspecting the semaphore from time to time doing something useful in between. Disadvantage is that the value is not stable: you cannot assert that the value is 0 after you have tested that.

Trace:

- sequence of atomic actions, obtained by interleavings of concurrent parts
- actions are assignments or tests

Invariant:

- special assertion that holds at every control point

Annotation:

- assertions at control points

Assertion:

- predicate (boolean function) in terms of the program variables
    - e.g. x=y, x≥0

• Control point:

- place in the program text where one might inspect the current state (i.e., values of variables) – between atomic actions
- Meaning: “when the program is started in a state that satisfies the initial assertion then, when a control point is reached during execution, the corresponding assertion holds”

Atomic action: finest grain of detail, indivisible

- sufficient at program level: single reference to shared variable in statement

Parallel execution: interleaving of atomic actions

Shared variables: accessible to several processes (threads)

Private variables: accessible only to a single process

Interference: disturbing others’ assumptions about the state

- usually, caused by “unexpected” interleavings
- particularly difficult with shared memory

Signaling disciplines:

- signal & exit
  - signal and exit the monitor immediately
  - sigall not supported
- signal & continue
  - Any processes released by this signal compete with the other processes to botain monitor access again
- signal & wait
  - singaller waits to access the montor again after the signal and competes with all other processes again
- signal & urgent wait
  - same as normal wait only the signaller has priority over other users.

Blocking:

- waiting for a condition to become true

Starvation:

- opposite of fairness: unpredictable, or infinite blocking times

Livelock:

- trying to pass a critical condition repeatedly without making any progress

Deadlock:

- extreme case of starvation: continuation not possible

We call a task (process or thread) blocked if:

- it is waiting on a blocking synchronization action
- or has terminated (including termination due to technical reasons)

A set D of tasks (with at least one non-terminated task) is called deadlocked if

- all tasks in D are blocked, and
- for each non-terminated task t in D, any task that might unblock t is also in D

Program behaviors that might lead to deadlocks:

- mutual exclusion
- greediness: hold and wait
- absence of preemption mechanism
- circular waiting

Analysis to see if a set is a deadlocked set is done by reduction, if what is left is a knot then this knot is the deadlocked set D.

Memory Management

A good memory is one that…

- …is non-volatile
- …is private for a program (protected)
- …has infinite capacity
- …has zero access time
- …is simple to use
- …is cheap

Memory hierarchy:

- CPU register memory
  - fast access (1 clk cycle)
  - but limited in size
- Primary (executable memory)
  - direct access (a few clk cycles),
  - relatively larger
- Secondary memory (storage devices)
  - I/O operations required for access,
  - very cheap / very large,
  - data stored for a longer period of time,
  - access speed is slow (3 orders of magnitude slower than CPU register memory)

Classical memory managers provides:

- Abstraction
  - MM (virtually) appears to be larger than the physical machine mem.
  - Memory as a large array of contiguously addressed bytes
  - Abstract set of logical addresses to reference physical memory
- Allocation
  - Allocate primary memory to processes as requested
- Isolation
  - Enable mutually exclusive use of memory by processes
- Memory Sharing
  - Enable memory to be shared by many processes

Adress binding:

- Instead of a set of physical primary memory addresses, processes are allocated a set of logical primary memory addresses
  - process address space
- Memory manager has to bind logical primary memory addresses to physical primary memory locations

When to bind:

- **Static**, i.e., during translation of program source to executable load
- module, or upon activation of a process (loading)
  - at programming time
  - at compile time
  - at link time
  - at load time
- **Dynamic**, i.e., during execution of the instructions of a running process (as late as possible).

Partitioning:

- Fixed partitions
  - Disadvantage: entire program is stored in MM until the end of execution
  - Disadvantage: internal fragmentation possible
- Dynamic partitions:
  - Advantage: No memory waste due to internal fragmentation
  - Disadvantage: memory waste not solved completely
  - Disadvantage: programs still have to be stored contiguously
  - Disadvantage: external fragmentation

Allocation schemes:

- First Fit
- Next Fit (Rotating First-Fit)
- Best Fit
- Worst Fit
- Optimized Schemes

Relocatable dynamic partitions:

- Compacting
  - When to compact?
  - Compacting creates overhead
- Swapping
  - When a program must be loaded into MM but there is not enough room so something has to be moved out to the secondary storage to make room

Virtual Memory is an abstraction for programs so they don't have to directly deal with the physical memory and finding a spot in the memory that fits.

Approach:

- Split the memory up in segments and try to fit parts of the program into these parts

Terminology:

- If segments are of different sizes: segmentation
- If segments are of the same size: paging
  - Physical memory blocks: (page)frames
  - Logical memory blocks: pages
- Paging and segmentation can be combined

Paging memory allocation:

- Advantage: an empty page frame is always usable by any process
- Advantage: compaction scheme is not required
- Disadvantage: mechanism needed to keep track of page locations

Translation to physical address from the frame table:

- TLB (translation look aside buffer)
  - Keeps track of the most recently used pages in MM so it's faster than to translate an address directly
    - Page not there? → page fault
    - Page fault is handled by page fault handler
      - what is done is decided by the policy for page removal
  - Problem with swapping things out of the memory is Trashing
    - When you spent more time swapping than actually executing

Replacement strategies:

- Min replacement (looks to the future)
- Random replacement
- FIFO replacement
- LRU replacement (Least frequently used)
- Clock replacement (second chance)
  - Circular list of all resident pages equipped with a use-bit u
  - Upon each reference u is set to 1 (set to 2 for third chance)
  - Search clockwise for u=0 while setting the use-bits to zero

Local replacement strategies:

- VMIN replacement (looks to the future)
  - At each memory reference
    - If there is a page fault then the requested page is loaded immediately
    - If that page is not referenced during the next τ memory references is removed
- Working Set (WS) model
  - Uses recent history (past τ memory references)
  - At each memory reference a WS is determined
    - Only the pages that belong to the WS reside in MM
    - A process can run if its entire WS is in MM
  - The working set is given by $W(t, \tau) = \{ r_j \mid t-\tau \leq j \leq t\}$
    - Reference string $r_j\ r_0\ r_1\ r_2 \ldots\ r_\tau$

Global replacement with static paging:
Replace the pages of which process? (depends on scheduling)

- Lowes priority process
  - Follows CPU scheduling (unlikely to be immediately scheduled again)
- Last process activated
  - Considered to be the least important
- Smallest process
  - Least expensive to swap out
- Largest process
  - Frees the largest number of frames

Load control:

- Which and how many pages to load
  - Static paging
    - Upon activation all pages of the process are loaded
  - Dynamic paging
    - Upon a fault one or more pages are loaded

Segmentation:

- MM is not divided into page frames because size of each segment is different
  - Memory allocated dynamically
- Each process has three obvious candidates
  - Code
  - Stack
  - Data
  - Other candidates are: stack of individual threads of a process, memory-mapped files

Segmentation with paging:

- Divide the segments into pages and use it in that manner
  - Virtual address contains s, p, w = segment, page, offset
  - Multimedia applications favor small number of segments big number of pages.

Cache memory:

- Contains information it's likely to be referenced to by the CPU, if we have a hit it will significantly improve the access time. If we fail it will increase the access time of course. We want to have the hit ratio as high as possible.

File Systems

Having many small files lead to internal fragmentation in file systems. Give two ways of solving this problem.

- Add the file content in the file descriptor.
- Use contiguous allocation for small files.
- Use blocks of variable size.

What is the working set, and what determines the lower and upper limits to its size?

- Working set is the set of physical memory pages currently dedicated to a running process. It includes the current top of stack, areas of the heap being accessed, the current area of code segment being executed, and any shared libraries recently used.
- Upper limit to its size: degree of multiprogramming.
- Lower limit: avoiding occurrence of page faults.

Which (possibly conflicting) requirements determine the choice of the page-size in a virtual memory system?

- It has to be a power of 2
- It has to be a big value so the page-table takes up a small amount of the memory.
- It has to be a low value to decrease the amount of internal fragmentation

Explain the difference between mounting and symbolic linking.

- symbolic link: name in new name system + closure are the contents of a file
- mounting: closure and prefixing are (transparently added to the resolution algorithm – joining two name spaces.

Task of file system:

- persistent across shutdowns
- independent of (virtual) memory size
- facilitates sharing between users, processes (not necessarily concurrent)
- Opening and closing files
    - Keep record in OFT (open file table)

it requires:

- naming
- transparency + portability (hide hardware details for this)
- robustness (against faults)
- access control (authorization, authentication)
- security (protection of user's files, data integrity)

User view concerns

- what constitutes a file
- naming of files
- allowed operations on files

OS implementation view concerns

- how files are structured
- how to keep track of free storage
- how many sectors are in a logical block

What is a file?

- According to user:
  - A named logical group of related data for:
    - Access control
    - Retrieval
    - Modification
- 2 options:
  - 1 File are just named byte (bit) sequences
    - The file system is unaware of file content & structure
    - Interpretation by application
  - 2 Files are structured based on their types
    - The file system is aware of file contents & structure
    - The type determines its use (interpretation and structure)
      - Editor can't open binary file, text file cannot be loaded for execution
      - Additional structure is included within the file, based on this type
- A file has attributes:
  - Type
  - Ownership
  - Size
  - Permanent/temporary flags
  - Protection/access rights
  - Dates

Access methods file:

- Sequential Access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind
  - convenient when medium was tape
- Direct Access
  - bytes/records read in any order
  - essential for database systems
  - read can be …
    - move current position (seek), then read
    - or given a start position, read read read…
- Access through index files
  - built on top of direct-access method
  - involves the construction of an index of the file
    - first search the index
    - then use the pointer to access the file directly

What really is a file system:

- Data structure on storage device that represents a collection of files
  - The description and documentation of this data structure
  - One might call this the physical file system
- OS subsystem that deals with files (named resources in general)
  - For files, it follows the mentioned rules to make them available via the API
  - This API is often called the virtual file system
  - It can deal with many different physical File Systems

Different kind of directories:

- Single-level directory
  - All files contained in a single directory
    - Naming problem
- Two-level directory
  - Separate directory for each user
    - More efficient searching
    - Still no grouping possibility for files of a user
- Tree-structured directories
  - Grouping possible
  - Efficient searching starting from current directory
- Graph directories
  - Acyclic (no cycles)
    - Shared subdirectories and files
  - General (cycles are allowed (be careful with searches))

Extended view of naming:

- Unified naming
  - Devices are seen as files as well

Per file it is listed who can do what to a file:

- Owner (Read, Write, Execute)
- Group (Read, Write, Execute)
- Public (Read, Write, Execute)
- A file could for instance have the access code 761, meaning:
  - 1, 1, 1 Owner
  - 1, 1, 0 Group
  - 0, 0, 1 Public

Operating System view – Implementation: 3 level structure:

- Directory level
- Basic file system level
- Device organization level

File organization on disk:

- Contiguous
  - Not flexible due to external fragmentation
- Linked
  - Simple linked blocks
    - Bad performance as random access is difficult
  - Using a dedicated table of pointers, one per block (the 'next' pointer, pointing to the next block)
  - Improved by linking adjacent blocks instead of linking individual blocks
- Indexed
  - Start with a block of block indices
    - File descriptor contains just the index of this block
  - Or a linked list of these index blocks (multilevel hierarchy)
  - Remember we also have the 2-level incremental index and the 3-level incremental index
    - Single, double or triple indirect blocks

We have to maintain a file of free blocks so that they can be used later:

- Bitmap, much alike a page table
- A ‘closure’ file storing all unused blocks

Remaining issues in the mapping to disk:

- Reliability
  - Errors:
    - Loss of a block
    - Inconsistencies
  - Dealing with it: replicated data
    - Repeat the basic tables
    - Store several versions
  - How to recover from errors
    - Journaling
- Efficiency
  - Organize the data so as to have the most efficient access

Journaling:

- Small circular buffer
- Task: write the operation to be performed before actually doing it
- On recovery read the journal to find recent operations and check if those operations actually completed, perform the operations that did not complete

Distributed file system:

- Goals:
  - Common naming
  - Hide the distributed structure
  - Support sharing via sharing semantics
- Issue: server performance:
  - Availability
    - Probability server is accessible
  - Reliability
    - Expected time to server failure
  - Scalability
    - Ability to grow a Distributed File System with number of clients as well as file-access request
  - Access times
- Naming types:
  - Global vs Local: is there a single directory structure seen by all?
  - Or can each user have its own view
- Location transparency
  - Does the path name reveal the hosting machine?
- Location independence
  - Must the path name change when a file is moved between machines

Sharing semantics options:

- Same as on single machine with multiple processes
  - effectuate writes 'as soon as possible' (visible)
  - problem: unpredictable delays
- Value/Result or session semantics
  - make copy upon open; write upon close
  - possibly combined with reservation
- Transaction semantics
  - reduce the time until changes are visible
  - atomic modifications, under application control
- Immutable-files
  - all files are read-only (modification impossible)
  - writing a file creates a new one à version management problem

Implementing a Distributive File System:

- Choices: Design choices concern the level where the connection client-server is put:
  - Application:
    - client just accesses the remote file system as just another application
      - needs a server application on the other side comparable to web-servers
  - Basic file system
    - client uses the remote file system as a process, with OFT's, file descriptors (references)
  - Device organization
    - client uses the remote file system as a block storage (and has to maintain its own OFT)

I/O Systems

Mention three reasons to use buffering inside the kernel.

- Sharing
- Caching
- Solve speed differences
- Solve user/kernel problems

What is the asynchronous part of a device driver?

- the part that is triggered by the device, viz., the interrupt handler.

Goals:

- to present a logical/abstract view of I/O devices
- to facilitate sharing of devices
- to provide efficiency in using devices (optimize performance)
  - CPU and multiple I/O devices execute concurrently.

'Classical classification':

- Block-oriented (storage) devices
  - Secondary storage
- Stream-oriented devices
  - Input and output characters (bytes) in sequential way
  - Keyboard
- Network communication devices
  - Send and receive packets

We have 2 kinds of IO:

- Low level IO
  - Drivers
- High level IO
  - API
  - Problem: 'classical classification not enough':
    - 1: memory-mapped I/O
    - 2: Direct access to driver
    - 3: extend the classes
      - Like Direct x

IO-buffering:

- Problems that it solves:
  - Process must wait for relative slow I/O to complete
  - Application may expect data in smaller pieces than a block
  - Virtual address range must remain in physical memory
  - Risk of a single process deadlock
- The buffer is just a dedicated memory range or Hardware register
- Main task:
  - Resolve speed difference and latency problems
  - Resolve granularity differences
  - Resolve problems with user and kernel space
- 2 types:
  - FIFO buffers
  - Direct access buffers
- Motivation:
  - Efficiency
    - Gamble on temporal locality
    - Gamble on spatial locality
    - Delay output
- Different kind of buffers:
  - No buffer
  - Single buffer
  - Double buffer
  - Circular buffer
- Every kind of buffer has metric-specifications:
  - Throughput
  - Latency

Disk scheduling:

- Rotation the disk and moving the arm is mechanical and slow, therefor we need good algorithms to make it more efficient:
  - FIFO
  - Shortest Seek First
  - Scan
  - C-Scan
  - Look
  - C-Look